IF184401 Design & Analysis of Algorithm IUP Midterm Exam
Lecturer : Misbakhul Munir Irfan Subakti, S.Kom., M.Sc.

Name : Faraihan Rafi Adityawarman
NRP : 5025211074

1. Create `isBST()`
    - **Problem description**

      Based on the file given, there's MyTree.java and MyTreeOps.java, we are needed to create a BST function to check whether the tree is BST or not.
    - **Program explanation**

      the helper function is used to see if the tree is truly a BST, by using a Boolean inside. To find it, we need to see the minimum and the maximum value of the tree by using `MIN_VALUE` and `MAX_VALUE` then using it to declare the `lowerBound` and `upperBound` integer.
    - **Source code**

```java
// 1. isBST() [20 points]
public static boolean isBST(MyTree t) {
    // Your code is in here
    if(isBST(t, Integer.MIN_VALUE, Integer.MAX_VALUE)) {
        return true;
    }
    return false;
}

// Helper function for isBST
// Get a boolean value to know whether 't' is BST (Binary Search Tree)
// whose values are within the range between lowerBound and upperBound
private static boolean isBST(MyTree t, int lowerBound, int upperBound) {
    // Your code is in here
    if (t == null)
        return true;

    //to check if there's already the same value in BST
    if(t.value < lowerBound || t.value > upperBound)
        return false;

    if(isBST(t.right, t.value + 1, t.value - 1)) {
        if(isBST(t.right, t.value + 1, upperBound)) {
            return true;
        }
    }

    return false;
}
```

2. Create `printDescending()`
    - **Problem description**

      We are needed to create a recursive function with the name `printDescending()` which takes an input from the BST t, then we print it with a sorted list in a descending order.

    - **Program explanation**

      The program function is used to print the value in descending order. By using functions inside `printDescending()` function, we can use it to find the reverse Inorder, by using `t.getRight()` and `t.getLeft()` after. But this is checked if the BST is filled or not, If it's empty, then it will return nothing, else it will print the descending order of the value.

    - **Source code**

```java
// 2. printDescending() [10 points]
public static void printDescending(MyTree t) {
    // Your code is in here
    if(t.getEmpty() == true)
        return;
    else {
        // this functions the same way as how inorder works
        // but instead of going inorder, it goes on reverse Inorder
        printDescending(t.getRight());
        System.out.println(t.getValue()+" ");
        printDescending(t.getLeft());
    }
}
```

3. Create `max()`
    - **Problem description**

      We are to make an efficient recursive function with the name `max()` with an input of BST t, to find the max value.

    - **Program explanation**

      This function works by traversing the BST from the right. It will search for the tree limit / until there is no more node left, then the loop will stop and the node will have the max value, then we return this value.

    - **Source code**

```java
// 3. max() [10 points]
/**
 * You have to: - handle empty trees - never look at left - never compares
 * values, i.e., the value of t and the right, because it's not necessary if
 * it's BST. - returns the right value as soon as found
 *
 * @param t is the tree being searched for the max value
 * @return the max value of tree t
 */
public static int max(MyTree t) {
    // Your code is in here
    MyTree current = t;
    if(t.getEmpty() == true)
        return 0;

    while(current.right != null)
        current = current.right;

    // return the value of max
    return(current.value);
}
```

4. Create `isHeightBalanced()`
    - **Problem description**

      We are to create a function to check if the height of tree is balanced or not by using the function `isHeightBalanced()` from the input MyTree t.

    - **Program explanation**

      The function `isHeightBalanced()` is checking to see if the tree is balanced. If it is, then it will return true, otherwise it will return false. The code starts by getting the left and right heights of the tree from `MyTreeOps.height()`, then it compares these two values with each other and see if that they are equal or greater than 1. If both are balanced then we return true; otherwise, we return false because there was a height imbalance between its left and right sides which would make it not be balanced overall.

- **Source code**

```java
// 4. isHeightBalanced() [10 points]
public static boolean isHeightBalanced(MyTree t) {
    // Your code is in here


    int leftHeight = MyTreeOps.height(t.left);
    int rightHeight = MyTreeOps.height(t.right);
    int generalHeight = leftHeight - rightHeight;

    // Checks if the tree has any empty nodes
    if(t.getEmpty()) {
        return true;
    }
    // Checks if the left and right height is balanced
    else if(generalHeight <= 1 && isHeightBalanced(t.left) && isHeightBalanced(t.right)) {
        return true;
    } else return false;

}
```

5. Create `InsertHB()`
   - **Problem description**

     We all know that an AVL Tree is supposed to be balanced, this is why we need to use a height balancing function to make this BST balance in a way. To make it that way, I need to create `InsertHB()` as a function with rebalancing constructor inside of it.
   - **Program explanation**

     The function `insertHB()` functions by inserting a new node into the tree, then using recursion until it reaches the base case that is `n < t.value` or `n > t.value`. If either one is true, then it will rebalance its children using a function called `rebalanceForLeft()` or `rebalanceForRight()`, respectively. The value of the local variable val is actually used to return the value of `rebalanceForLeft` and `rebalanceForRight` so I think it's a problem with my IDE, I'm sorry for the inconvenience.
   - **Source code**

```java
// 5. insertHB() [10 points]
public static MyTree insertHB(int n, MyTree t) {
    // Your code is in here

    //inserting node inside tree
    MyTree val;    The value of the local variable val is not used

    //checking if the tree is empty
    // if it's empty, then it'll return a new instance of MyTree with n
    if(t.getEmpty())
        return(new MyTree(n));

    //checking whether the tree has reached the base case
    if(n < t.value)
        return val = rebalanceForLeft(new MyTree(t.value, insertHB(n, t.left), t.right));
    else if(n > t.value)
        return val = rebalanceForRight(new MyTree(t.value, t.right.right, insertHB(n, t.left.left)));
    else return t;
}
```

6. Create `private static MyTree rebalanceForLeft(MyTree t)`
    - **Problem description**

        We are needed to create a function called `rebalanceForLeft(MyTree t)` so that we can check on the tree whether it's unbalanced on the left side or not.

    - **Code explanation**

        The function `private static MyTree rebalanceForLeft(MyTree t)` is used to rebalance the tree on the left. Adding +1 to the tree height of t.right is to check if the height is >= to the tree height of t.left, then the function can return t if the condition is met, but if it does not then, the function has to rebalance the function by creating a new subtree with the order of the nodes are changed to be balanced with the opposite side of the tree.

    - **Source code**

```java
// 6. rebalanceForLeft() [15 points]
private static MyTree rebalanceForLeft(MyTree t) {
    // Your code is in here

    // this will check if the tree is already balanced or not for the left side
    if(MyTreeOps.height(t.left) <= (MyTreeOps.height(t.right) + 1))
        return t;
    // if it's unbalanced, then it will balance it
    else {
        MyTree left = t.left;
        MyTree right = t.right;

        if(MyTreeOps.height(left.left) > MyTreeOps.height(right)) {
            MyTree val = new MyTree(left.value, left.left, new MyTree(t.value, left.right, right));
            return val;
        } else {
            MyTree val = new MyTree (left.right.value, new MyTree(left.value, left.left, left.right.left),
                        new MyTree(t.value, left.right.right, t.right));
            return val;
        }
    }
}
```

7. Create `private static MyTree rebalanceForRight(MyTree t)`
    - **Problem description**

      We are needed to create a function called `rebalanceForRight(MyTree t)` so that we can check on the tree whether it's unbalanced on the right side or not.
    - **Code explanation**

      The function `private static MyTree rebalanceForRight(MyTree t)` is used to rebalance the tree on the left. Adding +1 to the tree height of t.left is to check if the height is >= to the tree height of t.right, then the function can return t if the condition is met, but if it does not then, the function has to rebalance the function by creating a new subtree with the order of the nodes are changed to be balanced with the opposite side of the tree. It's basically the same as `private static MyTree rebalanceForLeft(MyTree t)` but instead of rebalancing the left side, it rebalance the right.

    - **Source code**

```java
// 7. rebalanceForRight() [15 points]
private static MyTree rebalanceForRight(MyTree t) {
    // Your code is in here

    // this will check if the tree is already balanced or not for the right side
    if(MyTreeOps.height(t.right) <= (MyTreeOps.height(t.left)) + 1)
        return t;
    // if it's unbalanced, then it will balance it
    else {
        MyTree left = t.left;
        MyTree right = t.right;

        if(MyTreeOps.height(right.right) > MyTreeOps.height(left)) {
            MyTree val = new MyTree(right.value, new MyTree(t.value, left, right.left), right.right);
            return val;
        } else {
            MyTree val = new MyTree(right.left.value, new MyTree(t.value, t.left, right.left.left),
                        new MyTree(right.value, right.left.right, right.right));
            return val;
        }
    }
}
```

8. Create `deleteHB(MyTree t, int x)`
    - **Problem description**

      We are to make a function called `deleteHB()` that is used to delete x from t but still making the tree balanced.

    - **Code explanation**

      First thing we do is to check whether the tree is empty or not, if it's empty, then return the empty tree. Then we will check if x is greater or lesser than the value itself. By doing this, we will know what function will run, like if it's less than or equal than the value, then we will delete the right subtree and then rebalance the left side and also vice versa. In some cases, where both of this

function didn't work, we can rebalance right for both case by deleting the left subtree and adding `max()` as the new parent.

- **Source code**

```java
// 8. deleteHB() [10 points]
/**
 * Deletes the value 'x' from the given tree, if it exists, and returns the new
 * Tree.
 *
 * Otherwise, the original tree will be returned.
 */
public static MyTree deleteHB(MyTree t, int x) {
    // Your code is in here     }

    // if it's empty, then it'll return the value of current t
    if(t.getEmpty())
        return t;

    // check if value is more than t.value or less t.value
    if(x > t.value) {
        MyTree temp = deleteHB(t.right, x);
        // after we delete it, we need to rebalance the tree again
        MyTree val = rebalanceForLeft(new MyTree(t.value, t.left, temp));
        return val;
    }
    else if(x < t.value) {
        MyTree temp = deleteHB(t.getLeft(), x);
        // after we delete it, we need to rebalance the tree again
        MyTree val = rebalanceForRight(new MyTree(t.value, temp, t.right));
        return val;
    } else {

        MyTree val = rebalanceForRight(new MyTree(max(t.left), deleteHB(t.left, max(t.left)), t.right));
        return val;
    }
}
```

By the name of Allah (God) Almighty, herewith I pledge and truly declare that I have solved quiz 1 by myself, didn't do any cheating by any means, didn't do any plagiarism, and didn't accept anybody's help by any means. I am going to accept all of the consequences by any means if it has proven that I have done any cheating and/or plagiarism.

Surabaya, 25 October 2022

Faraihan Rafi Adityawarman
5025211074